数理解析研究所講究録
第 454 巻 1982 年 404-426

A LOGICAL BASIS FOR PROGRAMMING METHODOLOGY II

SATORU TAKASU

RESEARCH INSTITUTE FOR MATHEMATICAL SCIENCES, KYOTO UNIVERSITY

INTRODUCTION: In the previous paper ([45]), we have developed a uniform method for putting proofs of the verification conditions of a while-program together to form a proof of its quantified specification. In this paper, we extend the method to PASCAL programs and briefly describe how to use the method for program synthesis incorporating it with a proof-checker system having Gentzen's sequent calculus as its underlying logic.

1. MAIN CONSTRUCTION: Asserted Pascal Statements and Their Templets of Proofs

(0) We assume that a many sorted theory T=(L,AxmS) is given. Let P(x,y), Q(x,y) and R(x,y) stand for formulas in L, and x and y stand for the vectors of input and output variables respectively, possiblly occuring in those formulas. No other variables are free.

(1) An asserted assignment statement is of the form

$$\{P(x,y)\}\ y:=F(x,y)\ \{Q(x,y)\}$$

and its verification condition is $P(x,y) \supset Q(x,F(x,y))$, where F is a vector of function symbols in L. The templet for this statement is

$$\frac{P(a,b),\ \Gamma \to Q(a,\ F(a,b))}{P(a,b),\ \Gamma \to \exists\, y.Q(a,y)}\ .$$

(2) An asserted if-statement is of the form

$$\{P(x,y)\}\ \underline{\text{if}}\ t(x,y)\ \underline{\text{then}}\ B_1\ \underline{\text{else}}\ B_2\ \{Q(x,y)\}$$

where t(x,y) is a quantifier-free formula in L , and B_1 and B_2 are statements, and its verification conditions are those of

$$\{P(x,y)\wedge t(x,y)\}\ B_1\ \{Q(x,y)\}\ \text{and}\ \{P(x,y)\wedge\sim t(x,y)\}\ B_2\ \{Q(x,y)\}.$$

The temlet for this statement is

$$\frac{\to t(a,b)\vee\sim t(a,b)\ ;\ \dfrac{t(a,b),\ P(a,b),\ \Gamma \to\exists\, y.Q(a,y)\ ;\ \sim t(a,b),\ P(a,b),\ \Gamma \to\exists\, y.Q(a,y)}{t(a,b)\vee\sim t(a,b),\ P(a,b),\ \to \exists\, y.Q(a,y)}}{P(a,b),\ \Gamma \to \exists\, y.Q(a,y)}$$

(3) An asserted while-statement is of the form

$$\{P(x,y)\}\ \underline{\text{while}}\ \{Q(x,y)\}\ t(x,y)\ \underline{\text{do}}\ B\ \{R(x,y)\}$$

and its verification conditions are the formulas

$$P(x,y)\supset Q(x,y)\quad ,\quad Q(x,y)\wedge\sim t(x,y)\supset R(x,y)$$

and the verification conditions of the asserted statement

$$\{Q(x,y)\wedge t(x,y)\}\quad B\quad \{Q(x,y)\}.$$

We assume a variable k expressing the number of visits to the loop occurs in Q without loss of generality.

The templet for this onsists of the synthesis part:

$$\dfrac{\dfrac{\dfrac{P(a,b),\Gamma \to \exists w.Q(a,0,w)}{\begin{array}{l}P(a,b) \to \\ \exists w.Q(a,0,w) \vee \\ \exists y.R(a,y)\end{array}} \;;\; \dfrac{\dfrac{\dfrac{\to t(a,n,d) \vee \sim t(a,n,d) \;;\; \dfrac{\dfrac{t(a,n,d),Q(a,n,d),\Delta \to \exists w.Q(a,n+1,w)}{t(a,n,d),Q(a,n,d),\Delta \to \exists w.Q(a,n+1,w) \vee \exists y.R(a,y)} \quad \dfrac{\sim t(a,n,d),Q(a,n,d),\Pi \to \exists y.R(a,y)}{\sim t(a,n,d),Q(a,n,d),\Pi \to \exists w.Q(a,n+1,w) \vee \exists y.R(a,y)}}{t(a,n,d) \vee \sim t(a,n,d),Q(a,n,d),\Delta,\Pi \to \exists w.Q(a,n+1,w) \vee \exists y.R(a,y)}}{Q(a,n,d),\Delta,\Pi \to \exists w.Q(a,n+1,w) \vee \exists y.R(a,y)}}{\exists w.Q(a,n,w),\Delta,\Pi \to \exists w.Q(a,n+1,w) \vee \exists y.R(a,y)} \;;\; \dfrac{\exists y.R(a,y) \to \exists y.R(a,y)}{\exists y.R(a,y) \to \exists w.Q(a,n+1,w) \vee \exists y.R(a,y)}}{\exists w.Q(a,n,w) \vee \exists y.R(a,y),\Delta,\Pi \to \exists w.Q(a,n+1,w) \vee \exists y.R(a,y)}}{P(a,b),\Gamma,\Delta,\Pi \to \exists w.Q(a,c,w) \vee \exists y.R(a,y)}}{P(a,b),\Gamma,\Delta,\Pi \to \forall k.(\exists w.Q(a,k,w) \vee \exists y.R(a,y))}$$

and combined with the following right part called the termination part:

$$\dfrac{\text{Synthesis Part} \;;\; \dfrac{\dfrac{\dfrac{Q(a,K,e),\theta \to \exists y.R(a,y)}{\exists w.Q(a,K,w),\theta \to \exists y.R(a,y)} \;;\; \exists y.R(a,y) \to \exists y.R(a,y)}{\exists w.Q(a,K,w) \vee \exists y.R(a,y), \theta \to \exists y.R(a,y)}}{\forall k.(\exists w.Q(a,k,w) \vee \exists y.R(a,y)),\theta \to \exists y.R(a,y)}}{P(a,b), \Gamma, \Delta, \Pi, \theta \to \exists y.R(a,y)}$$

(4) An asserted compound statement is of the form

$\{P(x,y)\}$ begin $B_1;B_2;\ldots;B_n$ end $\{R(x,y)\}$

and its verification conditions are those of

$\{P(x,y)\}$ begin $B_1;\ldots;B_{n-1}$ end $\{Q(x,y)\}$ and $\{Q(x,y)\}$ B_n $\{R(x,y)\}$.

Its templet is

$$\dfrac{P(a,b),\Gamma \to \exists w.Q(a,w) \;;\; \dfrac{Q(a,c),\Pi \to \exists y.R(a,y)}{\exists w.Q(a,w),\Pi \to \exists y.R(a,y)}}{P(a,b), \Gamma, \Pi \to \exists y.R(a,y)}$$

These (1)-(4) has been studied in Takasu (1981). The following examples exhibit how to use (1)-(4) for the synthesis of programs and also give a preliminary observation for the study of procedure statement and procedure declaration:

Example 1:(Quotient and Remainder)

Specification: For a pair of natural numbers a and non zero b, there exist the natural numbers q and r such that a=b*q+r and b>r .

To prove this specification, we use the following lemma:

Lemma: If b>0, then there holds for any k either (case 1) there exists R such that a=b*k+R, or (case 2) there exist q and r such that a=b*q+r and b>r .

The specification follows from the lemma (Termination Part):

To prove this, it suffices proving that case 1 implies the specification: If we take a+1 for k, then a=b*(a+1)+R never holds since b>0 implies b*(a+1)>a . This false-hood implies the specification. (The enough number of visits a+1, makes the loop invariant (case 1) false and the specification is attained by case 2; this is the termination.)

To prove the lemma, we use induction on k (Synthesis Part):

Initial step (k:=0;): We take a for R (R:=a;) so that case 1 holds for k=0.

Inductive step: (a) Case 2 for k implies case 2 for k+1. (b) Assume case 1 for k, namely a=b*k+R: We then consider the subcases with respect to R≥b v b>R . (Case R≥b) (while R≥b do begin k:=k+1;) Take R of k+1-th step to be R-b (R:=R-b) so that a=b*k+R implies a=b*(k+1)+R-b. (Case b>R) (end;) If b>R, then case 2 can be proved by taking k for q (q:=k;) and R for r (r:=R;).

We observe the relation between the above proof of the specification and the following program:

```
begin k:=0; R:=a; while R≥b do begin K:=k+1; R:=R-b end;
      q:=k; r:=R
end
```

Example 2: (GCD) To specify a program to compute the gcd of two natural numbers, we define the following predicates: $y|x \equiv \exists z.x=y*z$ and $GCD(a,b,z)=z|a \wedge z|b \wedge \forall y.(y|a \wedge y|b \supset y|z)$. We also assume the following propositions:

Proposition: (i) $\forall u. \forall v.(GCD(a,b,u) \wedge GCD(a,b,v) \supset u=v)$.

(ii) $\forall u.(GCD(b,a \bmod b,u) \supset GCD(a,b,u))$.

(iii) GCD(a,0,a).

Specification: For a positive number a and non negative number b, there exists z such that GCD(a,b,z).

Lemma: If a is positive and b is non negative, then for any non negative k, either (case 1) there exist a positive number x and a non negative number y such that $max(a,b)-k \geq y$ and GCD(x,y,u) implies GCD(a,b,u) for any u, or (case 2) there exists z such that GCD(a,b,z).

The specification follows from the lemma: First we take max(a,b) for k. (case 1) We take x for z of the specification and also x for u. Then we have y=0 which implies GCD(x,y,x). So GCD(a,b,z) in case 1 implies GCD(a,b,z) of the specification. Note that max(a,b) is the enough number of visits to the loop and the loop invariant exactly implies GCD(a,b,z).

We prove the lemma by induction on k:

Initial step : We take a for x and b for y in the lemma in the case of k=0 (x:=a; y:=b;). Then the lemma is clear since case 1 trivially holds.

Inductive step: The case 2 for k and the case 2 for k+1 are identical statements. Now we assume the case 1 for k. Then we make the case analysis with respect to y=0 v y≠0 :

(Case y≠0) (<u>while</u> y≠0 <u>do</u> <u>begin</u>) : For this case, we prove the case 1 for k+1 from case 1 for k: Let x and y denote themselves for k-th step and X and Y for k+1-th step. Now we take y for Y and x mod y for Y (r:=x mod y; x:=y; y:=r <u>end</u>;). Then we verify

i) x>0, y≥0 and y≠0 implies y>0 and x mod y ≥ 0.

ii) max(a,b)-k≥y> x mod y implies max(a,b)-(k+1) ≥ x mod y.

iii) GCD(y,x mod y, u) implies GCD(x,y,u).

(Case y=0) For this case we prove that the case 1 for k implies the case 2 for k+1. We take x for z in case 2, then to be proved is GCD(x,0,x) (z:=x;).

Then we have the following program:

```
begin     x:=a; y:=b;
     while y≠0 do begin r:=x mod y; x:=y; y:=r
                    end ; z:=x;
end
```

<u>Example 3</u>: (Factorial) We consider to compute the factorial of an non negative integer as follows:

<u>Axiom</u>: (i) isfact(0,1). (ii) For any non negative integer u, v, and x, isfact(x,v) and u=(x+1)*v imply isfact(x+1,u).

<u>Specification</u>: If n is non negative, then there exists r such that isfact(n,r).

<u>Lemma</u>: For any non negative k, either (Case 1) there exists a non negative m such that i) n≥ m=n-k ≥0 and ii) for any u, isfact(m,u) implies there exists r satisfying isfact(n,r*u), or (Case 2) there exists z such that isfact(n,z).

The specification follows from the lemma: We take n+1 for k, then the false-hood of m≥0 implies the specification and Case 2 implies also the specification.

We prove the lemma by induction on k:

Initial step: We prove Case 1 for k=0. We take n for m and 1 for r, then isfact(n,u) implies isfact(n,1*u) (begin M:=n; r:=1;).

Inductive step: We separate the cases with respect to m=0 v m>0 where this m is the one in k-th step.

(m>0): (while m>0 do begin) We prove Case 1 of the k+1-th step. We take m-1 for m of the k+1-th step (m:=m-1;). Then m>0 and n-k=m imply n-(k+1)=m-1. We take m*u for u of the k-th step, then isfact(m-1,u) implies isfact(m,m*u) by Axiom (ii). So there remains to prove that if isfact(n,r*m*u), then there exists r such that isfact(n,r*u). We take r*m for this r (r:=r*m end).

(m=0): We prove Case 2 of the k+1-th step. We take 1 for u and r for z of the k+1-th step (z:=r;). Then m=0 and isfact(0,1) imply isfact(m,1), and isfact(n,r*1) implies isfact(n,r).

Thus obtained program is

begin m:=n; r:=1; while m>0 do begin r:=m*r; m:=m-1 end end;

(5) An asserted procedure statement is of the form

{P(w,e)} q(w,e) {R(w,e)}

and its verification condition is

$$P(w,e) \supset \quad U(w,e) \wedge \forall u.(W(u,e) \supset R(u,e)) \qquad \text{(Adaptation)}$$

{U(u,v)} procedure q(u;v); B {W(u,v)}

is the corresponding procedure declaration, u is the vector of the non-local variables which subject to change in the body, v is the vector of the rest of the non-local variables, w is a part of the program variables and e is a vector of expressions.

(5-1) <u>Non-recursive case</u>: The templet for non-recursive procedure statement is the following:

$$\frac{R(d,e) \to R(d,e)}{W(d,e) \to W(d,e) \;;\; R(d,e) \to \exists w.R(w,e)}$$

$$\frac{W(d,e) \to W(d,e) \;;\; R(d,e) \to \exists w.R(w,e)}{W(d,e), W(d,e) \supset R(d,e) \to \exists w.R(d,e)}$$

$$\frac{W(d,e), W(d,e) \supset R(d,e) \to \exists w.R(d,e)}{W(d,e),\ \forall u.(W(u,e) \supset R(u,e)) \to \exists w.R(w,e)}$$

$$\frac{W(d,e),\ \forall u.(W(u,e) \supset R(u,e)) \to \exists w.R(w,e)}{U(c,e) \to U(c,e) \;;\; \exists u.W(u,e),\ \forall u.(W(u,e) \supset R(u,e)) \to \exists w.R(w,e)}$$

$$\frac{U(c,e) \to U(c,e) \;;\; \exists u.W(u,e),\ \forall u.(W(u,e) \supset R(u,e)) \to \exists w.R(w,e)}{U(c,e) \supset \exists u.W(u,e),\ U(c,e),\ \forall u.(W(u,e) \supset R(u,e)) \to \exists w.R(w,e)}$$

Specification of the procedure declaration

$$\frac{\Gamma,\ U(c,r) \to \exists u.W(u,r)}{\Gamma \to U(c,r) \supset \exists u.W(u,r)}$$

$$\frac{\Gamma \to U(c,r) \supset \exists u.W(u,r)}{\Gamma \to \forall v.(U(c,v) \supset \exists u.W(u,v))}$$

$$\frac{U(c,e) \supset \exists u.W(u,e),\ U(c,e),\ \forall u.(W(u,e) \supset R(u,e)) \to \exists w.R(w,e)}{\forall v.(U(c,v) \supset \exists u.W(u,v)),\ U(c,e),\ \forall u.(W(u,e) \supset R(u,e)) \to \exists w.R(w,e)}$$

$$\frac{\Gamma \to \forall v.(U(c,v) \supset \exists u.W(u,v)) \;;\; \forall v.(U(c,v) \supset \exists u.W(u,v)),\ U(c,e),\ \forall u.(W(u,e) \supset R(u,e)) \to \exists w.R(w,e)}{U(c,e),\ \forall u.(W(u,e) \supset R(u,e)),\ \Gamma \to \exists w.R(w,e)}$$

$$\frac{U(c,e),\ \forall u.(W(u,e) \supset R(u,e)),\ \Gamma \to \exists w.R(w,e)}{U(c,e) \wedge \forall u.(W(u,e) \supset R(u,e)),\ \Gamma \to \exists w.R(w,e)}$$

Verification condition

$$\frac{P(c,e), \Gamma \to U(c,e) \wedge \forall u.(W(u,e) \supset R(u,e)) \;;\; U(c,e) \wedge \forall u.(W(u,e) \supset R(u,e)),\ \Gamma \to \exists w.R(w,e)}{P(c,e)\ ,\ \Gamma \to \exists w.R(w,e)}\ .$$

(5-2) <u>Recursive case</u>:

Let K be an integer variable not appearing in the procedure body B. We put a virtual assignment statement "K:=K+1;" in front of the body B and the initial value of K is set to be -1 when the procedure is called up at a virtual place outside of the declaration. The value of K is specific for that body for each recursive call, namely, if the control is in the body for the first time, then K=0 and the recursive call in this body invokes another copy of the body and we have K=1 in this copy. The value of K is called the recursion depth of each stage.

Let u_K and v_K be the values of parameters at the time when the virtual assignment statement K:=K+1 is executed, and $g(u_K,v_K)$ a non-negative integer valued function satisfying $g(u_K,v_K)>g(u_{K+1},v_{K+1})$ (strictly decreasing function with respect to the recursion depth). Now we consider

$$SPEC \equiv \forall u.\forall v.(U(u,v)\wedge k=g(u,v)\geq 0 \supset \exists z.W(z,v))$$

as the specification of the recursive procedure declaration. To prove the specification, we first apply the course-of-value induction, namely we may assume

$$COURSE \equiv \forall h.(k>h \supset \forall u.\forall v.(U(u,v)\wedge h=g(u,v)\geq 0 \supset \exists z.W(z,v)))$$

always during the proving of the specification. Therefore, at the outset we have

$$\dfrac{\dfrac{U(c,d), k=g(c,d)\geq 0,\ COURSE \to \exists z.W(z,d)}{COURSE \to U(c,d)\wedge k=g(c,d)\geq 0 \supset \exists z.W(z,d)}}{COURSE \to \forall u.\forall v.(U(u,v)\wedge k=g(u,v)\geq 0 \supset \exists z.W(z,v)).}$$

Now, the templet for recursive procedure statements is as follows:

$$\dfrac{\dfrac{\begin{array}{l} U(a,e), k0=g(a,e)\geq 0 \\ \quad \to U(a,e)\wedge k0=g(a,e)\geq 0;\end{array} \quad \dfrac{\dfrac{\dfrac{\dfrac{R(\alpha,e)\to R(\alpha,e)}{W(\alpha,e)\to W(\alpha,e);\ R(\alpha,e)\to \exists w.R(w,e)}}{W(\alpha,e), W(\alpha,e)\supset R(\alpha,e) \to \exists w.R(w,e)}}{W(\alpha,e),\ \forall u.(W(u,e)\supset R(u,e))\to \exists w.R(w,e)}}{\exists z.W(z,e),\ \forall u.(W(u,e)\supset R(u,e))\to \exists w.R(w,e)}}{\begin{array}{l} U(a,e)\wedge k0=g(a,e)\geq 0 \supset \exists z.W(z,e), U(a,e), k0=g(a,e)\geq 0, \\ \quad \forall u.(W(u,e)\supset R(u,e)) \to \exists w.R(w,e)\end{array}}}{\begin{array}{l} \forall u.\forall v.(U(u,v)\wedge k0=g(u,v)\geq 0 \supset \exists z.W(z,e)), k0=g(a,e)\geq 0, \\ \quad U(a,e),\ \forall u.(W(u,e)\supset R(u,e)) \to \exists w.R(w,e)\end{array}}$$

$$\dfrac{\begin{array}{l} k>k0=g(a,e) \\ \to k>k0 \end{array} \qquad \begin{array}{l} \forall u.\forall v.(U(u,v)\wedge k0=g(u,v)\geq 0 \supset \exists z.W(z,e)), k0=g(a,e)\geq 0, \\ \quad U(a,e),\ \forall u.(W(u,e)\supset R(u,e)) \to \exists w.R(w,e)\end{array}}{\quad}$$

(continues to the next page)

$$\frac{k>k0 \supset \forall u.\forall v.(U(u,v) \wedge k0=g(u,v) \geq 0 \supset \exists z.W(z,v)), U(a,e), \quad k>k0=g(a,e) \geq 0, \ \forall u.(W(u,e) \supset R(u,e)) \rightarrow \exists w.R(w,e)}{U(a,e) \wedge k>k0=g(a,e) \geq 0 \wedge \forall u.(W(u,e) \supset R(u,e)), \quad \forall h.(k>h \supset \forall u.\forall v.(U(u,v) \wedge h=g(u,v) \geq 0 \supset \exists z.W(z,v))) \rightarrow \exists w.R(w,e)}$$

Verification Condition

$$\frac{\begin{array}{ll} P(a,e), k=g(c,d) \geq 0 & \exists k_0.\ (U(a,e) \wedge k>k_0=g(a,e) \geq 0 \wedge \forall u.(W(u,e) \supset R(u,e))), \\ \rightarrow \exists k_0.\ (U(a,e) \wedge & \forall h.(k>h \supset \forall u.\forall v.(U(u,v) \wedge h=g(u,v) \geq 0 \supset \exists z.W(z,v))) \\ k>k_0=g(a,e) \geq 0 \wedge & \rightarrow \exists w.R(w,e) \\ \forall u.(W(u,e) \supset R(u,e)); & \end{array}}{P(a,e), k=g(c,d) \geq 0, \ \forall h.(k>h \supset \forall u.\forall v.(U(u,v) \wedge h=g(u,v) \geq 0 \supset \exists z.W(z,v))) \rightarrow \exists w.R(w,e)}$$

<u>Example 4</u>: We consider a procedure to compute the gcd of x and y:

<u>Specification</u>: For any y and x, if x is positive and y is non negative, then there exists z such that GCD(x,y,z) holds.

In this specification, we regard y as g(x,y,z). Therefore, we may assume that for any h less than y, if x is positive and h is non negative, then there exists z such that GCD(x,h,z) holds, applying the course-of-value induction on y. This assumption is called the inductive assumption.

Now we separate the cases with respect to y=0 v y>0 (<u>if</u> y=0 <u>then</u>).

(y=0): We take x for z in the conclusion (z:=x <u>else</u>). Then the conclusion is clear from Proposition (iii) in Example 2.

(y>0): For this case, we use Proposition (ii) as the adaptation (gcd(y, x mod y ,z)). So we take x mod y for h in the inductive hypothesis so that y>x mod y holds. Furthermore, we take y for x in the inductive hypothesis so that we may assume GCD(y,x mod y,z) holds. Thus we can now use Proposition (ii) to prove the conclusion.

This proof corresponds to the following procedure declaration:

```
procedure gcd(x,y:integer; var z:integer);
begin if y=0 then z:=x else gcd(y, x mod y ,z) end;
```

Example 5: We consider a procedure to compute m!.

Specification: For any non negative integer m, there exists z such that isfact(m,z) holds.

Now we regard m as g(m,z) so that we have to prove:

Lemma 0: Assumption: (i) m is a non negative integer. (ii) For any non negative integer h less than m, there exists Z such that isfact(h,Z) holds. Conclusion: There exists z such that isfact(m,z) holds.

To prove Lemma 0, we separate the cases with respect to m=0 v m>0 (if m=0 then).

(m=0): Combining the axiom isfact(0,1) and the fact m=0, we have isfact(m,1) taking 1 for z (z:=1 else) in the conclusion.

(m>0): Under m>0, we may take m-1 for h in the assumption (ii). Therefore we must prove that isfact(m-1,Z) and m>0 imply the existence of z satisfying isfact(m,z). To do this, we prove the following lemmas:

Lemma 1: (Adaptation) If m>0, isfact(m-1,Z) and m-1≥0 and for any t, isfact(m-1,t) implies isfact(m,m*t), then there exists w such that isfact(m,m*w).

Lemma 2: If there exists w such that isfact(m,m*w), then there exists z such that isfact(m,z).

Lemma 3: If m>0 and for any non negative u, v and x, isfact(x,v) and u=(x+1)*v imply (isfact(x+1,u), then for any t, isfact(m-1,t) implies isfact(m,m*t).

Lemma 1 is clear if we take Z for t and w (begin fact(m-1,w);).

Lemma 2 is clear if we take m*w for z (z:=m*w end).

Lemma 3 is clear if we take m*t for u, t for v and m-1 for x.

Thus we have the following procedure declaration:

```
procedure fact(m:integer; var z:integer);
var w:integer;
begin if m=0 then z:=1
            else begin fact(m-1,w); z:=m*w end
end
```

Example 6:(Quicksort) We consider the following procedure declaration:

```
procedure quicksort(var A:array [1..N] of integer;
                        m,n:integer);
var i,j:integer;
bigin if n>m then begin  partition(A,i,j,m,n);
                         quicksort(A,m,j);
                         quicksort(A,i,n)
                    end
end
```

The verification of this declaration was given by M. Foley and C.A.R. Hoare (1971). This is specified as

$$\mathrm{Concld}(A,A_0,m,n)=\mathrm{Sorted}(A,m,n)\wedge \mathrm{Perm}(A,A_0,m,n)$$

where

$$\mathrm{Sorted}(A,m.n)= \forall p.\forall q.(n\geq q\geq p\geq m\supset A[q]\geq A[p])$$

$$\mathrm{Perm}(A,A_0,m,n)= \exists\begin{pmatrix} m & m+1 & \cdots & n \\ i_1 & i_2 & & i_{n-m}\end{pmatrix}.\ A=A_0\begin{pmatrix} m & m+1 & \cdots & n \\ i_1 & i_2 & & i_{n-m}\end{pmatrix}$$

and the procedure "partition" is specified as

$$\mathrm{Partd}(A,i,j,m,n)\wedge \mathrm{Perm}(A,A_0,m,n)$$

where

$$\mathrm{Partd}(A,i,j,m,n)\equiv\ i>j\wedge\forall p.\forall q.(i>p\geq m\wedge n\geq q>j\supset A[q]\geq A[p]).$$

It is also known (ibid.) that to verify the body of quicksort, we have the following only one verification condition:

$$A=A_0 \supset \underline{\text{if}}\ n>m\ \underline{\text{then}}$$

$$\exists B.(A=B\ \ n>m\ \forall C.\forall i.\forall j.(\mathrm{Partd}(C,i,j,m,n) \wedge \mathrm{Perm}(C,B,m,n)$$

$$\supset \exists D.(C=D \wedge \forall E.(\mathrm{Sorted}(E,m,j) \wedge \mathrm{Perm}(E,D,m,j)$$

$$\exists F.(E=F \wedge \forall G.(\mathrm{Sorted}(G,i,n) \wedge \mathrm{Perm}(G,F,i,n)$$

$$\supset \mathrm{Sorted}(G,m,n) \wedge \mathrm{Perm}(G,B,m,n)\quad)))\ )))$$

$$\underline{\text{else}}\quad \mathrm{Sorted}(A,m,n) \wedge \mathrm{Perm}(A,A_0,m,n)$$

which is easily proved.

Now our quantified specification is

$$\forall k.\forall A_0.\forall m.\forall n.(k=n-m \geq 0 \supset \exists A.(\mathrm{Sorted}(A,m,n) \wedge \mathrm{Perm}(A,A_0,m,n)))$$

where $g(m,n,A_0)$ is $n-m$.

To prove the above quantified specification, we use the course-of-value induction, namely we are going to prove

$$\forall h.(k>h \supset \forall A_0'.\forall m^*.\forall n^*.(h=n^*-m^* \geq 0 \supset \exists A.(\mathrm{Sorted}(A,m^*,n^*) \wedge \mathrm{Perm}(A,A_0^*,m^*,n^*)))$$

$$\rightarrow \forall A_0.\forall m.\forall n.(k=n-m \geq 0 \supset \exists A.(\mathrm{Sorted}(A,m,n) \wedge \mathrm{Perm}(A,A_0,m,n))).$$

First, we make the case analysis of $k=n-m>0 \vee k=n-m=0$ ($\underline{\text{if}}$ $n>m$ $\underline{\text{then}}$).

(n>m): To explain the detail, we set the following abbreviation:

$$L7 \equiv E=F \wedge \forall G.(\mathrm{Concld}(G,F,i,n) \supset \mathrm{Concld}(G,B,m,n))$$

$$L5 \equiv C=D\ \ \forall E.(\mathrm{Concld}(E,D,m,j) \supset \exists F.L7)$$

$$L2 \equiv A_0=B\ \ n>m\ \forall C.\forall i.\forall j.(\mathrm{Partd}(C,i,j,m,n) \wedge \mathrm{Perm}(C,B,m,n) \supset \exists D.L5)$$

$$\mathrm{Course}(h,k) \equiv k>h \supset \forall A_0'.\forall m^*.\forall n^*.(h=n^*-m^* \geq 0 \supset \exists A.\mathrm{Concld}(A,A_0',m^*,n^*)).$$

Using the cut inference with $\exists B.L2$, we introduce $L2(\underline{B}/B)$ to the assumption, since $\exists B.L2$ can be proved easily. So we have

$$k=n-m>0, L2(\underline{B}/B), \forall h.\mathrm{Course}(h,k) \rightarrow \exists A.\mathrm{Concld}(A,A_0,m,n).$$

Since

$$n>m \rightarrow \exists C.\exists i.\exists j.(\mathrm{Partd}(C,i,j,m,n) \wedge \mathrm{Perm}(C,A_0,m,n) \wedge n \geq i > j \geq m)$$

is the specification of the procedure "partition", we may introduce $\mathrm{Partd}(\underline{C},\underline{i},\underline{j},m,n)\ \mathrm{Perm}(\underline{C},A_0,m,n)$ to the assumption so that the same formula in $\mathrm{L2}(\underline{B}/B)$ can be eliminated (<u>begin</u> partition(A,i,j,m,n);). We have

$$\exists D.\mathrm{L5}(\underline{B}/B,\underline{C}/C,\underline{i}/i,\underline{j}/j),\ \forall h.\mathrm{Course}(h,k),A_0=B,n\geq\underline{i}>\underline{j}\geq m,n-m=k>0$$
$$\rightarrow \exists A.\mathrm{Concld}(A,A_0,m,n).$$

We make two copies of $\forall h.\mathrm{Course}(h,k)$ by contraction inference for the purpose to process L5 and L7. Now to process L5, we take $j-m$ for h. Then

$$n\geq\underline{i}>\underline{j}\geq m,k=n-m \rightarrow k>\underline{j}-m$$

so we may take m for m^*, $\underline{C}$ for A_0', and $\underline{j}$ for n^*. Since $\underline{j}-m\geq\underline{j}-m\geq 0$ is clear, we introduce $\underline{A}_1$ for A in $\mathrm{Course}(\underline{j},k)$. We introduce $\underline{D}$ for D and we take $\underline{A}_1$ for E in $\mathrm{L5}(\underline{B}/B,\underline{C}/C,\underline{i}/i,\underline{j}/j)$ so we have

$$\mathrm{Concld}(\underline{A}_1,\underline{C},m,\underline{j}),\mathrm{Concld}(\underline{A}_1,\underline{D},m,\underline{j})\supset\exists F.\mathrm{L7}(\underline{B}/B,\underline{i}/i,\underline{A}_1/E),\underline{D}=\underline{C},A_0=\underline{B},$$
$$\forall h.\mathrm{Course}(h,k),n\geq\underline{i}>\underline{j}\geq m,n-m=k\rightarrow \exists A.\mathrm{Concld}(A,A_0,m,n)$$

Now we can cancel $\mathrm{Concld}(\underline{A}_1,\underline{D},m,\underline{j})$ by $\mathrm{Concld}(\underline{A}_1,\underline{C},m,\underline{j})$ and $\underline{D}=\underline{C}$ (the adaptation is cancelled: quicksort(A,m,j);).

Similarly, to process L7, we take $n-\underline{i}$ for h, $\underline{F}$ for F and A_0', $\underline{i}$ for m^* and n for $n^*$and we introduce $\underline{A}_2$ for A in $\mathrm{Course}(n-\underline{i},k)$. So we have

$$\mathrm{Sorted}(\underline{A}_2,\underline{i},n)\wedge\mathrm{Perm}(\underline{A}_2,\underline{F},\underline{i},n)\supset \mathrm{Sorted}(\underline{A}_2,m,n)\ \mathrm{Perm}(\underline{A}_2,\underline{B},m,n),$$
$$\mathrm{Sorted}(\underline{A}_2,\underline{i},n)\wedge\mathrm{Perm}(\underline{A}_2,\underline{F},\underline{i},n),A_0=B,\underline{A}_1=\underline{F}\rightarrow \exists A.\mathrm{Concld}(A,A_0,m,n)$$

Now we may cancel $\mathrm{Sorted}(\underline{A}_2,\underline{i},n)\wedge\mathrm{Perm}(\underline{A}_2,\underline{F},\underline{i},n)$ (quicksort(A,i,n) <u>end</u>) and we arrive at the stage to determine A in the conclusion to be $\underline{A}_2$.

This example is very typical for the non recursive procedure statement and recursive procedure statements.

$(n=m)$: For this case we take A_0 for A in the conclusion, so nothing to do for computation.

(6) <u>An asserted for-statement</u> is of the form

(i) {P} <u>for</u> i:=m <u>to</u> n <u>do</u> B {Q([m..n])}

or

(ii) {P} <u>for</u> i:=n <u>down</u> <u>to</u> m <u>do</u> B {Q([m..n])

and the corresponding verification conditions are

(i) $P \supset Q([m..m])$ and those of

$\{n \geq i \geq m \wedge Q([m..i))\}$ B $\{Q([m..i])\}$,

(ii) $P \supset Q([n..n])$ and those of

$\{n \geq i \geq m \wedge Q((i..n])\}$ B $\{Q([i..n])\}$

respectively. The corresponding templets are the followings:

(i)

$$\begin{array}{ll}
 & \Gamma, Q(a,[m..m+k],d), n\geq m+k+1 \to \exists y.Q(a,[m..m+k+1],y) \\ \hline
n\geq m+k+1 \to n\geq m+k; & \Gamma, \exists y.Q(a,[m..m+k],y), n\geq m+k+1\geq m \to \exists y.Q(a,[m..m+k+1],y) \\ \hline
 & \Gamma, n\geq m+k+1, n\geq m+k \supset \exists y.Q(a,[m..m+k],y) \to \exists y.Q(a,[m..m+k+1],y) \\ \hline
P(a,c),\Gamma \to n\geq m \supset \exists y.Q(a,[m..m],y); & \Gamma, n\geq m+k \supset \exists y.Q(a,[m..m+k],y) \to n\geq m+k+1 \supset \exists y.Q(a,[m..m+k+1],y) \\ \hline
\end{array}$$

$$\Gamma, P(a,c) \to n\geq m+(n-m) \supset \exists y Q(a,[m..m+(n-m)],y)$$

(ii)

$$\begin{array}{ll}
 & \Gamma, Q(a,[n-k..n],d), n-(k+1)\geq m \to \exists y.Q(a,[n-(k+1)..n],y) \\ \hline
n-(k+1)\geq m \to n-k\geq m; & \Gamma, \exists y.Q(a,[n-k..n],y), n-(k+1)\geq m \to \exists y.Q(a,[n-(k+1)..n],y) \\ \hline
 & \Gamma, n-(k+1)\geq m, n-k\geq m \supset \exists y.Q(a,[n-k..n],y) \\
 & \quad \to \exists y.Q(a,[n-(k+1)..n],y) \\ \hline
P(a,c),\Gamma \to & \Gamma, n-k\geq m \supset \exists y.Q(a,|n-k..n|,y) \\
n\geq m \supset \exists y.Q(a,[n..n],y); & \quad \to n-(k+1)\geq m \supset \exists y.Q(a,[n-(k+1)..n],y) \\ \hline
\end{array}$$

$$\Gamma, P(a,c) \to n-(n-m)\geq m \supset \exists y.Q(a,[n-(n-m)..n],y).$$

<u>Example 7</u>: We consider the following program:

<u>var</u> i,h:m..n; max:T; A:<u>array</u> <u>of</u> T;

<u>begin</u> h:=m; max:=A[m];

```
for i:=m to n do if A[i]>max then
                          begin max:=A[i] ; h:=i end
end
```

We set

$$Q([m..m+k],h,max)\equiv n\geq h\geq m \wedge max=A[h] \wedge \forall j.(m+k>j\geq m \supset max\geq A[j]);$$

then the specification of the above program can be expressed as

$$\exists h.\exists max.Q([m..n],h,max) \ .$$

To prove this specification, we use the induction on k:

Initial step: To prove $\exists h.\exists max.Q([m..m],h,max)$, we take m for h and A[m] for max (begin h:=m ; max:=A[m] ;). Then $Q([m..m],m,A[m])$ is clear.

Inductive step: To prove

$$Q([m..m+k],\underline{h},\underline{max}),n\geq m+k+1 \rightarrow \exists h.\exists max.Q([m..m+k+1],h,max)$$

we make the case analysis of $A[m+k+1]>max \vee max\geq A[m+k+1]$.

($A[m+k+1]>max$: (if A[i]>max then begin): For this case, we take m+k+1 for h and A[m+k+1] for max. Then $Q([m..m+k+1],m+k+1,A[m+k+1])$ can be proved easily from $Q([m..m+k],\underline{h},\underline{max})$ by making the case analysis of $j=m+k+1 \vee m+k\geq j\geq m$. (h:=i; max:=A[i] end).

($max\geq A[m+k+1]$): For this case, we take h for h and max for max. (Here we have no computation so that we have no else-part (end).)

(7) An asserted case-statement is of the form

$\{P\wedge(x\in[k_1,\ldots,k_n])\}$ case x of $k_1:S_1;\ldots;k_n:S_n$ end $\{Q\}$

where x is the selector , k_i ($n\geq i\geq 1$) a label and S_i a statement. The verification conditions for this statement are those of

$\{P\wedge x=k_i\}\ S_i\ \{Q\}$ for $n\geq i\geq 1$.

Its templet is

$$\cfrac{\cfrac{x=k_{n-1},P(a,c)\rightarrow\exists y.Q(a,y);x=k_n,P(a,c)\rightarrow\exists y.Q(a,y)}{x=k_{n-1}\vee x=k_n,\ P(a,c)\rightarrow\exists y.Q(a,c)}}{\cdot\ \cdot\ \cdot}$$

$$\frac{x=k_1,P(a,c)\rightarrow\exists y.Q(a,y);x=k_2\vee\ldots\vee x=k_n,P(a,c)\rightarrow\exists y.Q(a,y)}{x=k_1\vee x=k_2\vee\ldots\vee x=k_n,\ P(a,c)\rightarrow\exists y.Q(a,y)}\ .$$

(8) An asserted repeat-until-statement is of the form

$\{P\}$ repeat B until t $\{Q\wedge t\}$

where B is a statement and t is a quantifier-free formula in L. Its verification conditions are

$Q\wedge\sim t\supset P$

and those of

$\{P\}\ B\ \{Q\}$.

Since the above statement is equivalent to the statement

$\{P\}$ begin B; while $\{Q\}$ $\sim t$ do B $\{Q\wedge t\}$,

its templet can be similar to the templet for while-statement. But, the initial step of the induction used in the templet of while-statement must determine the same statement B as in the case of $\sim t$ at the inductive step. This implies that our system must check whether two parts of the proof are identical or not. By this reason, we exclude the repeat-until statements in our system together with the go-to-statements.

2. AN INTERACTIVE PROGRAM SYNTHESIZER:

Our proof-checker system interactively constructs proofs in many-sorted formal systems having a common syntactic rules to form terms and formulas and a common set of inference rules LJ+Induction. Each user must declare first a language (consisting of parameters, variables, function symbols and predicate symbols to be used) and a set of axioms. These materials are used to construct Pascal declarations automatically.

The declared function symbols and predicate symbols consist of those built in Pascal and/or those with the previously proved specifications being declared as axioms. The latter kind of axioms are associated with Pascal function and procedure declarations stored in a file.

After the language and axiom declarations, the interactive sessions take place for constructing a proof. At the outset, a sequent to be proved is supplied from the user, together with the informations whether the user regards the sequent as a specification of a program or of a procedure and which bound variables are regarded as output variables (or variable parameters of the procedure). Then the construction of the proof starts with this sequent upwards and right most first (with respect to the proof-tree), receiving some inference commands. Here we think, for each command, "to prove the conclusion of the inference, it suffices to prove the premisses" . The inference commands consist of, not only the primitive inferences in LJ+Induction, but also some compound inference commands. Each of these compound inference commands is associated with a templet of a proof which corresponds to a Pascal statement as we describe these in Section 1. Each of these compound inference commands constructs a part of the proof according to the corresponding templet and also it constructs a part of the program forming the corresponding statement in a top-down manner using some stackes and in the manner of Dijkstra's structured programming.

Besides the usage of the compound inference commands, the system keeps the track of term dependence for the purpose to detect the input variables (or value parameters) and the program variables (or local variables). If a term t is selected for an existentially quantified bound variable z (by an inference $\rightarrow\exists$), then the bound variables whose eigen parameters (introduced by $\exists\rightarrow$ or $\rightarrow\forall$) are occuring in t, are said to be dependent on the target variable z. Such dependent variables in the right branch of a cut or induction inference may be depended by some other variables in the left branch. So we may extend transitively the dependence relation over the entire proof. After the completion of the proof, the variables, dependent on the output variables and not having dependent variables, are input variables (or value parameters). The variables, dependent on the output variables and not being input variables, are program variables (or local variables).

At the end of the proving of a quantified specification, an executable Pascal program or a function or procedure declaration will remain in a file. If it is an executable program, then it can be fed into a Pascal compiler. The constructed program is always totally correct provided that the quantified specification was the right one.

REFERENCES

[1] Ackerman,W.: Zur Wiederspruchsfreikeit der Zahlentheorie, Math.Ann. 117(1940)162-194.

[2] Alagic,S. & Arbib,M.A.: The design of Well-Structured and Correct Programs, Springer-Verlag 1978.

[3] Bibel,W. & Schreiber,J.: Proof search in a Gentzen-like system of first-order logic, In E.Gelenbe & D. Potier edi., International Computing Symposium 1975 Amsterdam, North-Holland.

[4] Bledsoe,W.W.: A man-machine theorem proving system, Artificial Intelligence 5(1974)51-72.

[5] Bledsoe,W.W.:Non-resolution theorem proving, Artificial Intelligence 9(1977)1-35.

[6] Bledsoe,W.W. & Tyson,M.: The UT interactive theorem prover, Memo. ATP-17a, The Univ. of Texas at Austin, Math. Dept. 1978.

[7] Dahl,O.J. &Hoare,C.A.R.: Hierarchical program structures, In O.J. Dahl,E.W.Dijekstra & Hoare: Structured Programming, Academic Press (1972)197-220.

[8] Darlington,J.: Application of program transformation to program synthesis, In Colloques IRIA on Proving and Improving Programs, Arc et Senans,(1975)133-144.

[9] Darlington,J.: Synthesis of several sorting algorithms, Acta Informatica 11(1978)1-30

[10] Dijekstra,E.W.: A Disciplin of Programming, Prentice Hall, 1976.

[11] van Emden,M.H. & Kowalsky,R.A.: The semantics of predicate logic as a programming language, JACM 23(1976)23-32.

[12] Floyd,R.W.: Assigning meaning to programs, In Proc. of the Symp. in Appl. Math. vol. 19, AMS Providence (1967)19-32.

[13] Floyd,R.W.: Toward interactive design of correct programs,Proc. of IFIP Congress (1971)7-10, North-Holland.

[14] Foley,M. & Hoare,C.A.R.: Proof of a recursive program: Quicksort, Computer J. 14(1971)391-395.

[15] Gentzen,G.: Untersuchungen über das logische Schliessen, I, II, Math. Z. 39(1935)176-210.

[16] Gödel,K.: Über eine bischer noch nicht benützte Erweiterung des finiten Standpunktes, Dialectica 12(1958)76-82.

[17] Green,C.: The design of PSI program synthesis system, In Proc. of 2nd Int. Joint Conference on Software Engi.(1976) San Fransisco.

[18] Hoare,C.A.R.: Algorithm 65: FIND CACM 4(1961)321.

[19] Hoare,C.A.R.: The Axiomatic Basis of Computer Programming, CACM 12(1969)576-583.

[20] Hoare,C.A.R.: Proof of a program FIND, CACM 14(1971)39-45.

[21] Hoare,C.A.R.: Procedures and Parameters; an axiomatic approach, Symp. on Semantics of Algorithmic Language, (E. Engeler edi.) Lec. Note in CS 188 (1971).

[22] Hoare,C.A.R.: A note on the for statement,BIT 12(1972)334-341.

[23] Hoare,C.A.R. & Wirth,N.: An axiomatic definition of PASCAL, Acta Informatica 2(1973)335-355.

[24] Igarashi,S. et al.: Automatic program verification I: Logical basis and its implementation, Acta Informatica 4(1975)145-182.

[25] Kleene,S.C.: Introduction to Metamathematics, van Nostrand 1952.

[26] Kowalski,R.: Predicate logic as a programming language,Proc. of IFIP Congress (1974)569-574, Amsterdam, North-Holland.

[27] Kowalski,R.: Logic for Problem Solving, 1979 North-Holland.

[28] Kreisel,G.: Some uses of proof theory for finding computer programs, Colloque Int. des Logique (1977)123-134, Paris.

[29] Luckhum,D. & Nilsson,N.J.: Extracting Information from resolution

proof trees,Artificial Intelligence 2(1971)

[30] LuckhumD.C.: Program Verification and Verification-Oriented Programming , Proc. of IFIP Congress (1977)783-793, North-Holland.

[31] Manna,Z. & Waldinger,R.: Toward automatic program synthesis, CACM 14(1971)151-165.

[32] Manna,Z. & Waldinger,R.: Knowledge and reasoning in program synthesis, Artificial Intelligence 6(1975)175-208.

[33] Manna, Z. & Waldinger,R.: Studies in Automatic Programming Logic, North-Holland 1977.

[34] Manna, Z. & Waldinger,R.: Synthesis: Dreams - Programs, IEEE Trans. on Software Engineering 5(1979)294-328.

[35] Manna,Z. & Waldinger,R.: A deductive approach to program synthesis, Proc. of 6-th IJICAI (1979)542-551 Tokyo.

[36] McCarthy,J.: Problems in the theory of computation, Proc. of IFIP Congress (1965)219-222, W.A.Kalenich edi., Spartan Book Co.

[37] Mendelson,E.: Introduction to Mathematical Logic, D. van Nostrand, 1964.

[38] Nauer,P.: Proofs of algorithms by genersl snapshots BIT (1966)310-316.

[39] Sato,M.: Toward a mathematical theory of program synthesis, Proc. of the 6-th IJICAI (1979)575-762.

[40] Schütte,K.: Proof Theory, Springer-Verlag, 1977.

[41] Smullyan,R.: First-Order Logic, Springer-Verlag 1968.

[42] Suzuki,N.: Automatic verification of programs with complex data structures, Ph.D. Thesis 1976 Stanfor Univ., published also in Outstanding Dissertations in the Computer Science, Garland Publ. Co. 1980.

[43] Szabo,M.E. edi.: The collected papers of Gerhard Gentzen, North-Holland Publ. Co.

[44] Takasu,S.: Proofs and Programs, in Proc. of the 3rd IBM Symp. on Math. Foundations of Computer Science 1978, IBM Japan.

[45] Takasu,S. & Kawabata,S.: A logical basis for programming methodology, Theoretical Computer Science 16(1981)43-60.

[46] Takeuti,G.: Proof Theory, North-Holland 1975.

[47] Troelstra,A.S.: Metamathematical Investigation of Intuitionistic Arithmetic and Analysis, Lec. Note in Math. No.344, 1973.

[48] Tyugu,E.H.: A programming system with automatic program synthesis, Lec. Note in Computer Science No. 47, Methods of Algorithmic Language Implementation, (1977)251-267.

[49] Tyugu,E.H.: Towards Practical Synthesis of Programs, Proc. of IFIP Congress (1980) S.H.Levington edi.,North-Holland Publ. Co.

[50] Waldinger,R. & R.C.T.Lee: PROW: A step tward automatic program writing, In Proc. of IJICAI (1969)241-252.